# Interactive Time-Dependent Particle Tracing Using Tetrahedral Decomposition

David N. Kenwright, David A. Lane

Abstract- Streak lines and particle traces are effective visualization techniques for studying unsteady fluid flows. For real-time applications, accuracy is often sacrificed to achieve interactive frame rates. Physical space particle tracing algorithms produce the most accurate results although they are usually too expensive for interactive applications. An efficient physical space algorithm is presented in this paper which was developed for interactive investigation and visualization of large, unsteady, aerodynamical simulations. Performance has been increased by applying tetrahedral decomposition to speed up point location and velocity interpolation in curvilinear grids. Preliminary results from batch computations [1] showed that this approach was up to six times faster than the most common algorithm which uses the Newton-Raphson method and trilinear interpolation. Results presented here show that the tetrahedral approach also permits interactive computation and visualization of unsteady particle traces. Statistics are given for frame rates and computation times on single and multi-processors. The benefits of interactive feature detection in unsteady flows are also demonstrated.

Index Terms- Particle tracing, scientific visualization, computational fluid dynamics, time-dependent, unsteady flow, streak line, curvilinear grid, tetrahedral decomposition.

## I. INTRODUCTION

Unsteady particle tracing is a relatively new visualization technique that has emerged because of the need to visualize unsteady or time-dependent data sets. Steady-state flow simulations only require one set of grid and solution data to describe a flow, while unsteady flow simulations may comprise of hundreds or thousands of time steps of data. Each time step has an associated grid and

solution file. The size of these data sets can run into hundreds of gigabytes [2,3].

Numerical techniques for visualizing unsteady flows mirror those used in experimental fluid mechanics and include path lines, time lines and streak lines.

- *path line*: generated by tracing the path of a single particle (also called a particle path).
- *time line*: generated by tracing a line of particles which are all released at the same time.
- *streak line*: generated by continuously injecting particles from a fixed location.

Streak lines, also called filament lines, are the most popular visualization technique and also the simplest to generate. In wind tunnels, they are produced by injecting smoke into flow [4]. Stream lines, curves that are tangential to a vector field, are not generally used to visualize unsteady flows because they do not show the actual motion of particles in the fluid but rather the theoretical trajectories of particles with infinite velocity. In steady flows, stream lines are identical to path lines and streak lines, but in unsteady flows they can differ significantly. For example, Fig. 1 shows stream lines, path lines, streak lines, and time lines computed from an unsteady simulation of flow around an oscillating airfoil. All techniques are shown at the 128th time step. The important flow feature in this data set was the vortex shedding caused by stalling. Both the streak lines and time lines revealed the vortices well but the stream lines and path lines did not. The latter techniques fail to capture these features because they do not show the actual motion of the fluid over time.

Fig. 1. Comparison of stream lines, path lines, streak lines and time lines in an unsteady flow.

The goal of this study was interactive computation and visualization of streak lines from large unsteady computational fluid dynamics (CFD) simulations. In computational flow visualization, streak lines are generated by releasing particles at discrete intervals, usually in accordance with the simulation time steps. The continuous injection of particles leads to a rapid growth of the number of particles in the flow, all of which must be traced until they leave the flow field or until the simulation ends. There may be several thousand active particles in an unsteady flow, so it is essential that the advection or tracing process be as efficient as possible.

## II. PREVIOUS WORK

Many algorithms have been presented for particle tracing in steady flows yet relatively few consider the extension to unsteady flows. This extension is not trivial because the time varying nature of the flow and grid adds complexity to almost every part of the algorithm [3]. Consequently, unsteady particle traces are usually computed in a batch process and then played back upon completion [5,6,7,8]. A significant problem with this approach is that the particle injection points must be chosen in advance. Usually, an excessive number of particles are released (>10,000) to prevent missing important flow structures. A better approach is to interactively position the streak line injectors and study the particle motion in real time.

Interactive unsteady particle tracing was achieved by Bryson and Levit [9] by simplifying the algorithm and data set. They transformed the velocity vector field into a uniform (computational) space to make the numerical integration simpler, and resampled the grid to enable the entire data set to be stored in memory. However, many of the subtle features in the flow were lost as a result. Particle tracing in computational space has recently come under scrutiny [10] and was shown to have poor accuracy compared with physical space schemes.

## III. PHYSICAL VS COMPUTATIONAL SPACE TRACING

Body-fitted or curvilinear grids are widely used to model the complex geometries of aerospace vehicles. Some CFD flow solvers internally transform these curved grids into a uniform Cartesian

space, usually called computational space, to make numerical calculations simpler and more efficient. On output, the solution data is generally transformed back to the physical grid space for post analysis and visualization.

As with flow solvers, particle tracing algorithms may operate in both computational and physical space. Computational space schemes require the curvilinear grid, and the associated velocity field, to be remapped into computational space where the particles are then advected. When the mapping is done as a preprocessing step for the whole grid, this makes particle tracing very efficient [9].

The main disadvantage of tracing in computational space is that the transforming Jacobian matrices are usually approximations, so the transformed vector field may be discontinuous. Also, if there are irregularities in the grid, such as cells with collapsed edges, the transformed velocities may be infinite [11]. Analyses by Sadatjoen et al. [10] and Hultquist [12] in steady flows have shown that this mapping technique produces significant errors in distorted curvilinear grids. Calculating Jacobian matrices in unsteady flows with moving grids is more inaccurate because each matrix has three additional time-dependent terms which must be approximated [13].

> Remark: It would be preferable and more accurate if the computational space data were saved directly from the flow solver. However, there is no provision in the PLOT3D data file format, used widely by NASA, for storing these fields. This motivated the improvement of physical space particle tracing.

Physical space tracing schemes are preferred because the interpolation and integration processes are done on the curvilinear grid which eliminates the need to evaluate Jacobian matrices and transform the velocity field. The only disadvantage is that the task of locating particles is more complicated and hence more expensive. This problem is addressed in this paper. An explicit point location technique is presented which has yielded a significant speed-up over the most common point location technique: the Newton-Raphson iterative method.

## IV. PHYSICAL SPACE TRACING ALGORITHM

Physical space algorithms proceed by first searching out the element or cell which bounds a given point. This is termed the *cell search* or *point location* process. Once found, the velocity is evaluated at that point by interpolating the nodal velocities. In unsteady flows, the velocity components usually need to be interpolated temporally as well as spatially. This necessitates loading two or more time steps of data into memory. Intermediate positions of the grid may also have to be interpolated if the grid changes in time. The particle's path is determined by solving the differential equation for a field line:

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}(\mathbf{r}(t), t) \tag{1}$$

where $\mathbf{r}$ is the particle's location and $\mathbf{v}$ the particle's velocity at time t. Integrating (1) yields:

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \int_t^{t+\Delta t} \mathbf{v}(\mathbf{r}(t), t)\, dt \tag{2}$$

The integral term on the right hand side can be evaluated numerically using a multi-stage method (e.g., Runge-Kutta, Bulirsch-Stoer) or a multi-step method (e.g., backwards differentiation, Adams-Bashforth). Issues concerning the accuracy and stability of these methods are discussed by Darmofal and Haimes [14]. Regardless of how it is solved, the end result is a displacement which when added to the current position, $\mathbf{r}(t)$, gives the new particle location at time $t+\Delta t$.

The essential steps in a time-dependent particle tracing algorithm are as follows:

1. Specify the injection point for a particle in physical space, $(x,y,z,t)$.
2. Perform a point location to locate the cell that contains the point.
3. Evaluate the cell's velocities and coordinates at time t by interpolating between simulation time steps.
4. Interpolate the velocity field to determine the velocity vector at the current position, $(x,y,z)$.

5. Integrate the local velocity field using equation (2) to determine the particle's new location at time $t+\Delta t$.

6. Estimate the integration error. Reduce the step size and repeat the integration if the error is too large.

7. Repeat from step 2 until particle leaves flow field or until $t$ exceeds the last simulation time step.

It is important to note that step 5 may involve repeated applications of steps 2, 3, and 4 depending on which numerical integration scheme is used. The 4th order Runge-Kutta scheme used in this study actually requires three repetitions to advance from time $t$ to $t+\Delta t$. Refer to Section IV.E for more details.

### A. Point location in tetrahedral cells

The core problem in all particle tracers is: given an arbitrary point $\mathbf{x}$ in physical space, which cell does this point lie in and what are its natural coordinates. The natural coordinates, also called barycentric or computational coordinates, are local non-dimensional coordinates for a cell. Strictly, computational coordinates are different because they are globally defined.

The widely used trilinear interpolation function provides the opposite mapping to that required for point location, that is, it determines the coordinates of $\mathbf{x}$ from a given natural coordinate $(\xi,\eta,\zeta)$. Unfortunately, it cannot be inverted easily because of the non-linear products, so it is usually solved numerically using the Newton-Raphson method.

By using a point location technique based on tetrahedral elements, the natural coordinates can be evaluated directly from the physical coordinates. Tetrahedral elements permit the use of a linear interpolation function to map from natural to physical coordinates:

$$\mathbf{x}(\xi,\eta,\zeta) = \mathbf{x}_1 + (\mathbf{x}_2-\mathbf{x}_1)\xi + (\mathbf{x}_3-\mathbf{x}_1)\eta + (\mathbf{x}_4-\mathbf{x}_1)\zeta \qquad (3)$$

Note that $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ are the physical coordinates at the vertices of the tetrahedron (see Fig. 2). The natural coordinates $(\xi,\eta,\zeta)$ vary, as per usual, from 0 to 1 in the non-dimensional cell.

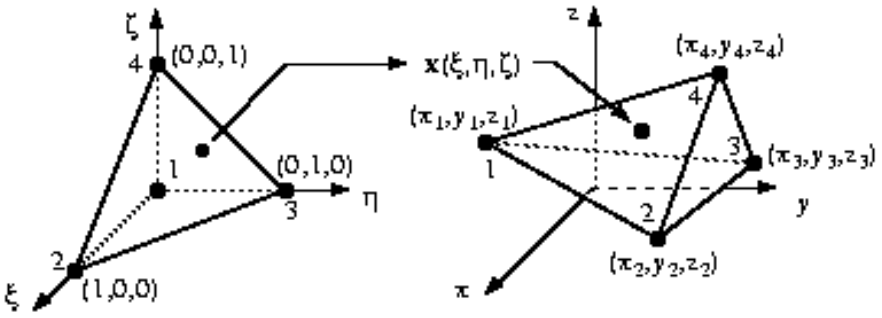


Fig. 2. Tetrahedron geometry in natural (non-dimensional) and physical coordinate spaces.

Equation (3) can be inverted analytically because it does not have any non-linear terms. The solution for the natural coordinates at a given physical point $(x_p,y_p,z_p)$ is given by:

$$\begin{pmatrix}\xi\\ \eta\\ \zeta\end{pmatrix}=\frac{1}{V}\begin{pmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{pmatrix}\begin{pmatrix}x_p-x_1\\ y_p-y_1\\ z_p-z_1\end{pmatrix} \tag{4}$$

The constants in the 3x3 matrix are:

$$\begin{aligned}
a_{11} &= (z_4-z_1)(y_3-y_4) - (z_3-z_4)(y_4-y_1)\\
a_{21} &= (z_4-z_1)(y_1-y_2) - (z_1-z_2)(y_4-y_1)\\
a_{31} &= (z_3-z_3)(y_1-y_2) - (z_1-z_2)(y_3-y_3)\\
a_{12} &= (x_4-x_1)(z_3-z_4) - (x_3-x_4)(z_4-z_1)\\
a_{22} &= (x_4-x_1)(z_1-z_2) - (x_1-x_2)(z_4-z_1)\\
a_{32} &= (x_2-x_3)(z_1-z_2) - (x_1-x_2)(z_2-z_3)\\
a_{13} &= (y_4-y_1)(x_3-x_4) - (y_3-y_4)(x_4-x_1)\\
a_{23} &= (y_4-y_1)(x_1-x_2) - (y_1-y_2)(x_4-x_1)\\
a_{33} &= (y_2-y_3)(x_1-x_2) - (y_1-y_2)(x_2-x_3)
\end{aligned}$$

and the determinant, V (actually 6 times the volume of the tetrahedron), is given by:

$$V = (x_2-x_1)[\,(y_3-y_1)(z_4-z_1)-(z_3-z_1)(y_4-y_1)\,] +$$
$$(x_3-x_1)[\,(y_1-y_2)(z_4-z_1)-(z_1-z_2)(y_4-y_1)\,] +$$
$$(x_4-x_1)[\,(y_2-y_1)(z_3-z_1)-(z_2-z_1)(y_3-y_1)\,]$$

The natural coordinates $(\xi,\eta,\zeta)$ can be evaluated in 104 floating point operations by implementing the equations above. This figure can be halved by precomputing common terms before evaluating the matrix coefficients and determinant.

**B. Tetrahedral decomposition**

The hexahedral cells in curvilinear grids must be decomposed into tetrahedra in order to use equation (4). Since the data sets for time-dependent flows are usually extremely large, it is impractical to do the decomposition as a preprocessing step. It must be performed on the fly as particles enter cells.

A hexahedral cell can be divided into a minimum of five tetrahedra (Fig. 3). This decomposition is not unique because the diagonal edges alternate across a cell. Since the faces of a hexahedron are usually non-planar, it is important to ensure that adjoining cells have matching diagonals to prevent gaps. This is achieved by alternating between an odd and even decomposition as illustrated in Fig. 3. In a curvilinear grid, the correct configuration is selected by simply adding up the integer indices of a specific node (the node with the lowest indices was used in practice). Choosing the odd configuration when the sum is odd and the even configuration when the sum is even guarantees continuity between cells.

> Remark: The odd and even configuration may be switched. This has no impact on the point location but it can affect the velocity interpolation because different vertices are used. In practice, because CFD grids have such a fine resolution, these differences are slight and have only been detected in regions with high velocity gradients.

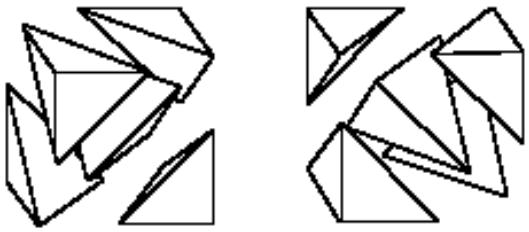

Fig. 3. Sub-division alternates between two configurations to ensure continuity between cells.

### C. Cell-search scheme

Equation (4) allows the natural coordinates to be evaluated directly from the physical coordinates with relatively little effort. There are four conditions which must be valid for the point $(x_p,y_p,z_p)$ to lie within the tetrahedron. They are:

$$\xi \geq 0 \;:\; \eta \geq 0 \;:\; \zeta \geq 0 \;:\; 1-\xi-\eta-\zeta \geq 0 \qquad (5)$$

If any one of these is invalid then the point is outside the tetrahedron. In particle tracing algorithms, this happens when particles cross cell boundaries. The problem then arises of which tetrahedron to advance to next. The solution is quite simple since the natural coordinates tell you which direction to move. For example, if $\xi<0$, the particle would have crossed the $\xi=0$ face. Similarly, if $\eta<0$ or $\zeta<0$, the particle would have crossed the $\eta=0$ or $\zeta=0$ face respectively. If the fourth condition is violated, i.e. $(1-\xi-\eta-\zeta)<0$, then the particle would have crossed the diagonal face. The cell-search proceeds by advancing across the respective face into the adjoining tetrahedron. The look-up tables needed to identify this tetrahedron are given in Appendix I.

Occasionally, two or more conditions in (5) may be violated if a particle crosses near the corner of a cell or if it traverses several cells at once. In such cases, the worst violator of the four conditions is used to predict the next tetrahedron. Even if the bounding tetrahedron is not the immediate neighbour, by always moving in the direction of the worst violator the search will rapidly converge upon the correct tetrahedron. Tests in numerous grids, including complex C-grids

with singulat points, have nevet tutned up a case whete this technique has failed to locate the bounding tettahedton.

The cell seatch ptocedute desctibed above should only be used if the cell being sought is neatby, that is, within a few cells of the ptevious one. This is usually the case duting patticle ttacing since the majotity of patticles only ctoss one cell at a time. Thete ate two situatiots when the cell being sought is not likely to be neatby, these being: i) at the statt of a patticle ttace and ii) aftet jumping between gtids in multi-zone data sets. In these citcumstances, the cell seatch should be pteceded by anothet scheme in otdet to ptevent weaving actoss a latge gtid one tettahedton at a time. We use the 'boundaty seatch' technique desctibed by Buning [15].

**D. Velocity interpolation in a tetrahedron**

In unsteady flows, the velocity field changes in time as well as space. Since the velocity fields ate only solved at disctete time steps and at disctete locations on the gtid, they must be intetpolated in both time and space. In the ptesent algotithm, these intetpolatiots ate handled sepatately.

D.1 Tempotal Intetpolation

The tempotal intetpolation is petfotmed fitst using a lineat function applied between the two closest time steps. Fot example, at a given time t which lies between time steps $t_l$ and $t_{l+1}$, the velocity $\mathbf{u}$ at an atbittaty gtid node (i,j,k) is given by:

$$\mathbf{u}^{t}_{i,j,k} = (1-\delta)\,\mathbf{u}^{t_l}_{i,j,k} + \delta\,\mathbf{u}^{t_{l+1}}_{i,j,k} \qquad (6)$$

whete the time ftaction $\delta$ is $(t-t_l)/(t_{l+1}-t_l)$. Note that equation (6) only evaluates the velocity at a given node, no spatial intetpolation is petfotmed at this stage. Note also that if the gtid moves in time, a tempotal intetpolation of the gtid positions is also tequited. We used a simnilat lineat intetpolation function fot this putpose. The tempotal gtid intetpolation must ptecede the point location, although the tempotal velocity intetpolation does not have to. It is, howevet, convenient

to perform both at once since the time fraction, $\delta$, is the same for both interpolants. These temporal interpolations are only applied locally to a single tetrahedron, that is, the current one being used by the point location or velocity interpolation procedures.

D.2 Spatial Interpolation

One of three techniques may be used for the spatial interpolation of velocity: physical space linear interpolation [16], volume weighted interpolation [15], and linear basis function interpolation [17]. All three are mathematically equivalent [1,18] and produce identical interpolation functions. The authors showed previously [1] that the linear basis function was the most efficient technique for this application because it reused the natural coordinates computed during point location. Using the numbering convention in Fig. 2, the linear basis function for spatial velocity interpolation is:

$$\mathbf{u}(\xi,\eta,\zeta) = \mathbf{u}_1 + (\mathbf{u}_2-\mathbf{u}_1)\xi + (\mathbf{u}_3-\mathbf{u}_1)\eta + (\mathbf{u}_4-\mathbf{u}_1)\zeta \qquad (7)$$

where $\xi$, $\eta$ and $\zeta$ are the natural coordinates computed in equation (4), and $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, and $\mathbf{u}_4$ are the velocity vectors at the vertices.

## E. Numerical Integration Scheme

The numerical integration of equation (2) was performed with a 4th order Runge-Kutta scheme. For a time-dependent flow with moving grid geometry, this takes the form:

$$\begin{aligned}
a &= \mathbf{v}(\mathbf{r}(t),t)\Delta t \\
b &= \mathbf{v}\left(\mathbf{r}(t)+\frac{a}{2},t+\frac{\Delta t}{2}\right)\Delta t \\
c &= \mathbf{v}\left(\mathbf{r}(t)+\frac{b}{2},t+\frac{\Delta t}{2}\right)\Delta t \\
d &= \mathbf{v}(\mathbf{r}(t)+c,t+\Delta t)\Delta t \\
\mathbf{r}(t+\Delta t) &= \mathbf{r}(t)+\frac{1}{6}(a+2b+2c+d)
\end{aligned} \qquad (8)$$

where $\mathbf{r}$ is the particle position, $\mathbf{v}$ the velocity vector at that position, and $\Delta t$ the time step. The four stages of the Runge-Kutta scheme span three time values ($t$, $t+\Delta t/2$, and $t+\Delta t$) and therefore require new grid and velocity data at each one. Fortunately, these only need to be interpolated in the tetrahedron that surrounds a particle. However, a particle moves in both time and space during integration, so it may lie in different tetrahedra during the four intermediate stages of equation (8). Point location and velocity interpolation must therefore be performed after every stage.

> Remark: There are numerical integration schemes which require fewer velocity evaluations than the Runge-Kutta scheme, e.g., the Bulirsch-Stoer method [19]. These may be substituted to further improve the performance of this unsteady particle tracing algorithm.

### F. Step size adaptation

If the integration step size is fixed at a constant value along the entire particle path, or regulated to achieve a specified number of steps per cell, a particle may understeer around bends if the flow changes direction rapidly. This can be prevented by using an adaptive step size control scheme where the integration step size is changed according to an error tolerance.

The error tolerance can be computed using a standard numerical technique such as *step doubling* [19] whereby a particle is advanced forward from a given point using a step size $\Delta t$ and then the process is repeated from the same point using two half steps of size $\Delta t/2$. The step size is reduced if the distance between the end-points is greater than a specified tolerance; a number usually deduced by trial and error. This technique was implemented and tested but was found to be too expensive for interactive particle tracing because it performed a large number of point locations.

A heuristic technique for adapting step size was suggested by Darmofal and Haimes [20]. They measured the angle between velocity vectors at successive points along a particle path to estimate the change in velocity direction. We implemented that scheme and a very similar one which adapted the step size according to the angle between successive line segments on a path line (Fig. 4).

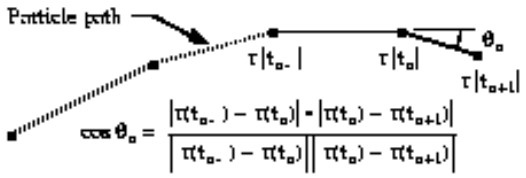


Fig. 4. Step size adaptation is based on the change in the path line direction.

Both of these schemes worked well in practice and ran approximately three times faster than the step doubling algorithm. In both cases, the initial step size was estimated using:

$$\Delta t = \frac{1}{|\mathbf{u}|}\sqrt[3]{\frac{V}{6}} \tag{9}$$

where $|\mathbf{u}|$ is the magnitude of the velocity at the current position and V is the determinant from equation (4). This ensures the initial particle displacement is commensurate with the length scale of the tetrahedron. Following this initial estimate, the step size is halved if the angle $\theta_n$ is too large ($\theta_n > 15°$), doubled if it is too small ($\theta_n < 3°$), or kept the same if it is in between these limits ( $3° \leq \theta_n \leq 15°$ ). Through experimentation in several data sets we found this scheme produced particle traces with the same accuracy as the step doubling scheme, based on an error tolerance of $10^{-5}$ in the latter, when adaptation angles of 15 degrees (upper limit) and 3 degrees (lower limit) were used.

## V. INTERACTIVE PERFORMANCE EVALUATION

In a previous paper by the authors [1], the performance and accuracy of the tetrahedral method was compared with a conventional particle tracing algorithm in batch computations of streak lines. The conventional algorithm, originally developed by Buning [15] for steady flows and extended to unsteady flows by Lane [8], used an iterative Newton-Raphson method for point location and trilinear functions for grid and velocity interpolations. The execution profiles confirmed that velocity interpolation and point location were the most computationally expensive tasks in the

conventional algorithm. With the tetrahedral method, computation times were improved by up to a factor of six while still maintaining the accuracy of the streak lines.

The tetrahedral method has since been implemented in C++ in the Virtual Windtunnel [9], an interactive visualization system for studying unsteady flows. Results presented here demonstrate the interactive performance of the tetrahedral method in that system.

The interactive tests were performed on an SGI Onyx with four 75MHz R8000 processors and five gigabytes of RAM. The large memory capacity enables moderately sized unsteady flow simulations to be loaded into physical memory, thus avoiding disk latencies when accessing the data. The tapered cylinder data set [7] was used for all the interactive tests (see Fig. 5). It had 100 simulation time steps and 131k grid points (grid dimensions 64 x 64 x 32). Each time step consisted of approximately 1.5 megabytes of velocity data.

Fig. 5. The tapered cylinder data set used for the interactive performance tests. The ten streak lines shown here could be computed and rendered at up to 33 frames per second.

To find important flow features in an unsteady flow, a large number of particles must be injected into the flow over time. In this respect, our objective was to determine how many streak lines could be generated interactively with the tetrahedral method.

Fig. 6 shows the aggregate time taken to compute and render streak lines, as a function of the number of streak lines, for 100 time steps. All computations were performed at run-time and did not use any precomputed fields. Fig. 6 shows there is a linear relationship between the number of streak lines and their computation time. It also shows a neat linear speed-up when computations are distributed over two or three processors. These linear trends occur because streak lines are comprised of discrete particles and can be advected independently. When running on multi-processors, the Virtual Windtunnel creates $p$-1 light weight processes, where $p$ is the number of processors, using SGI parallel programming primitives such as m_fork. One processor is always reserved by the Virtual Windtunnel for operating system and graphics tasks.

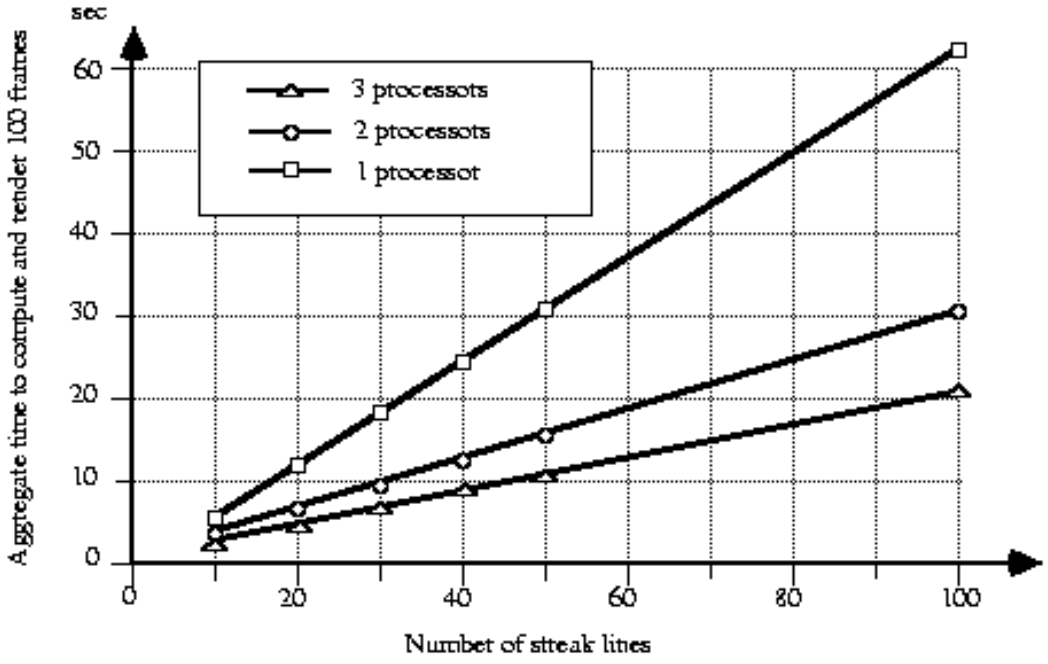


Fig. 6. Time taken to compute and render 100 frames as a function of the number of streak lines. Timings are given on one, two, and three processors.

The performance is expressed in terms of frame rate in Fig. 7. One frame corresponds to one simulation time step. A frame rate of 10 frames per second is the accepted baseline for interactive visualization [9]. The results in Fig. 7 show that up to 15 streak lines could be computed and rendered at this rate on one processor. Likewise, up to 32 streak lines could be computed and rendered on two processors and 44 streak lines on three processors at 10 frames/sec. It is important to note that the number of particles in a streak line increase linearly over time, so earlier frames are much quicker to compute than later ones. The frame rates shown here are averaged over 100 time steps. Higher frame rates could be obtained by reducing the number of time steps.

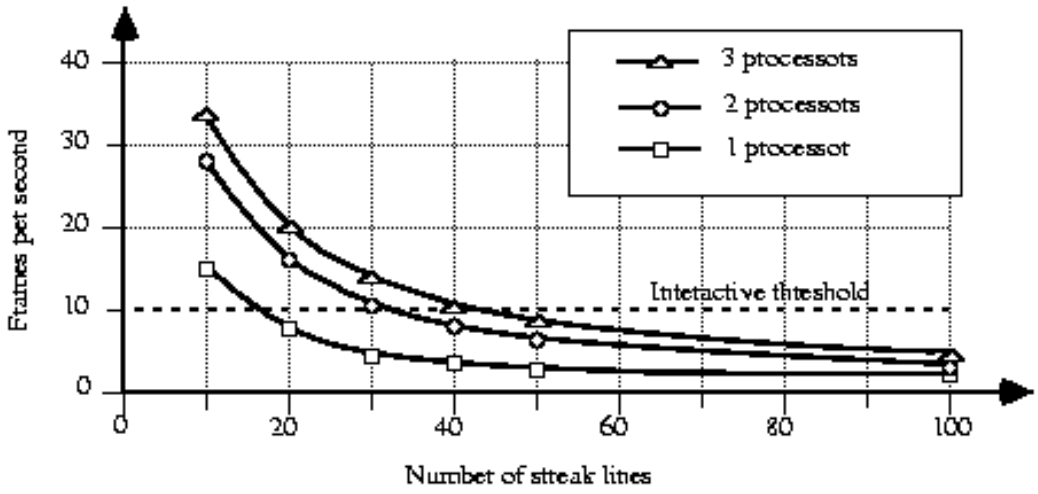


Fig. 7. Frame rates for computing and rendering streak lines on one, two, and three processors.

Another interesting metric is the particle advection rate. The total number of particle advections for m streak lines over n time steps is given by:

$$\text{number of advections} = m\sum_{j=1}^{n} j \qquad (10)$$

Dividing this by the time taken to compute the streak lines gives the particle advection rate. At the

interactive threshold (see Fig. 7), the advection rate is about 8k particles/sec on one processor, 14k particles/sec on two processors, and 22k particles/sec on three processors.

## VI. INTERACTIVE STREAK LINES AND FEATURE DETECTION

It was illustrated in Fig. 1 that particle paths of individual particles can give misleading results because they sometimes fail to detect important flow features. By injecting large numbers of particles we are more likely to capture these features, although actually seeing them can be difficult because particles clump together in clouds when there is a lot of mixing or recirculation in the flow.

We found it most useful to interactively probe the flow using a rake with 10 to 40 injectors. However, users are warned of a problem with interactive streak lines. Streak lines, unlike stream lines, take time to propagate, so the user should not move the rake of injectors too quickly. It is best to periodically stop and let the streak lines cycle through all the time steps before moving to another location.

The benefits of near real-time performance and interactive rake adjustments are demonstrated in Fig. 8. In this example, the user wanted to visualize the von Karman vortex shedding in the wake of the tapered cylinder. The general area of interest was first identified by using a wide rake with 20 injectors. The rake was then shortened and used to probe for a region of flow reversal. Once the desired location was found, the number of injectors was increased to make the streak lines more coherent and to highlight the vortex shedding. The particle colour depicts the velocity magnitude.

Fig. 8. Flow features can easily be detected in an unsteady flow by interactively moving the streak line injectors.

Interactive streak lines were also used to visualize the flow over a delta wing at a high angle of attack, as shown in Fig. 9. This data set had 100 simulation time steps and a single-zone grid like the tapered cylinder, but was over nine times larger. The velocity field for each time step was nearly 14 megabytes, and the entire data set nearly 1.4 gigabytes. Although this data set was significantly larger than the test case, the performance of the particle tracing was not degraded. This is because particle advection only requires local cell searches and interpolations. Previous results [1] have shown that there is a performance penalty on multi-zone grids because global searches are required when particles move into new grids.

Fig. 9. Both local and global flow structures were visualized by interactively changing the number of streak lines and the length of the rake in this unsteady flow around a delta wing.

## VII. FUTURE DIRECTIONS

The tapered cylinder and delta wing data sets used in this study are small compared to the multi-zone data sets currently being computed at NASA Ames Research Center. Only a small number of time steps from these large multi-zone data sets will fit into physical memory on even the largest workstations. Two approaches are being investigated to allow scientists to visualize more of their data. The first is by simply subsetting the grid to reduce its size and to localize the region of interest. The second is by using a large disk array with a high bandwidth. A disk array with 96 disks and 200 gigabytes of storage capacity is currently being evaluated for this application. Data transfer rates of over 300 megabytes per second have been measured. Potentially, this will enable us to read solution files with 30 megabytes of data per time step at interactive frame rates.

## VIII. CONCLUSIONS

Tetrahedral decomposition has made the point location and velocity interpolation tasks more efficient in a particle tracing algorithm for time-dependent flows. Streak lines were computed and rendered at interactive frame rates on an SGI Onyx workstation in a data set with 100 time steps. Almost linear scaling in performance was measured when one, two, and three processors were used. Up to 44 streak lines could be computed and visualized at 10 frames per second when the calculations were performed on three processors. This permitted interactive positioning and adjustment of streak line injectors which aided vortex detection in an unsteady flow.

# APPENDIX I

### A. Node Numbering Convention

Different node numbering conventions are used for the "odd" and "even" tetrahedral decompositions to simplify the look-up tables used for point location in Appendix I.C.

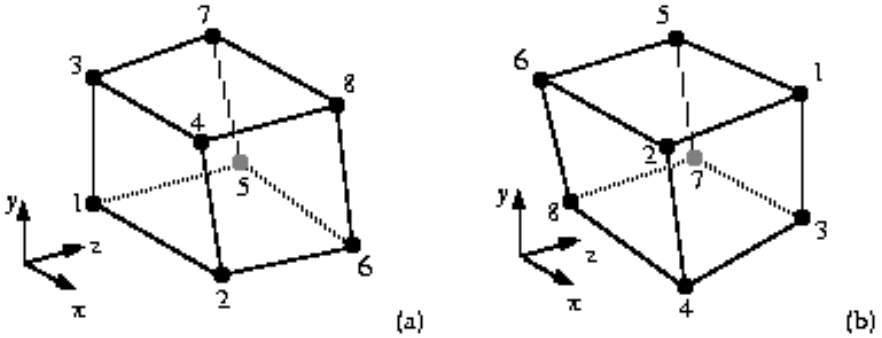


Fig. 10. The (a) odd and (b) even numbering conventions.

### B. Tetrahedral Decomposition

The following diagrams illustrate how a hexahedral cell is decomposed into 5 tetrahedra. Note that all tetrahedra are drawn in the natural coordinate system (refer to Fig. 2 for more details).

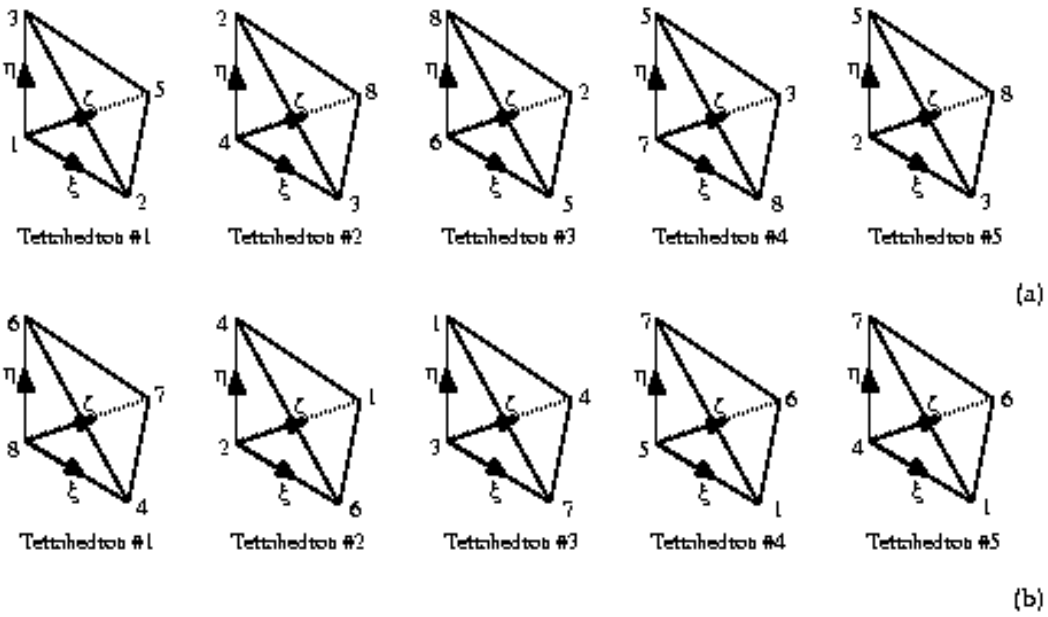


Fig. 11. The (a) odd and (b) even tetrahedral decompositions.

### C. Point location look-up tables

Two look-up tables are used to locate the tetrahedron that bounds a particle. The first table predicts which tetrahedron a particle will move into when it exits the current tetrahedron through a particular face. The exit face is determined from four conditional tests which are based on the natural coordinates $(\xi,\eta,\zeta)$, (refer to Section IV.C for more details). Note that this table applies to both the odd and even tetrahedral decompositions.

| Conditional test | Tetrahedron #1 | Tetrahedron #2 | Tetrahedron #3 | Tetrahedron #4 | Tetrahedron #5 |
|---|---|---|---|---|---|
| $\zeta < 0$ | #4 | #1 | #2 | #3 | #1 |
| $\eta < 0$ | #3 | #2 | #1 | #4 | #2 |
| $\xi < 0$ | #2 | #3 | #4 | #1 | #3 |
| $1-\xi-\eta-\zeta < 0$ | #5 | #5 | #5 | #5 | #4 |

Fig. 12. The adjacent tetrahedron look-up table.

A second table is used to update the cell indices (i,j,k) when a particle moves into a different hexahedral cell. Because each cell is divided into five tetrahedra, particles may cross several tetrahedra before they actually leave a cell. As with the previous table, the number of the current tetrahedron and an exit face are needed to identify the appropriate row and column. The entry "same cell" in this table indicates that the neighbouring tetrahedron is in the same hexahedral cell.

| Conditional test | Tetrahedron #1 | Tetrahedron #2 | Tetrahedron #3 | Tetrahedron #4 | Tetrahedron #5 |
|---|---|---|---|---|---|
| $\zeta < 0$ | k = k-1 | k = k-1 | k = k+1 | k = k+1 | same cell |
| $\eta < 0$ | j = j-1 | j = j+1 | j = j-1 | j = j+1 | same cell |
| $\xi < 0$ | i = i-1 | i = i+1 | i = i+1 | i = i-1 | same cell |
| $1-\xi-\eta-\zeta < 0$ | same cell | same cell | same cell | same cell | same cell |

(a)

| Conditional test | Tetrahedron #1 | Tetrahedron #2 | Tetrahedron #3 | Tetrahedron #4 | Tetrahedron #5 |
|---|---|---|---|---|---|
| $\zeta < 0$ | i = i+1 | i = i+1 | i = i-1 | i = i-1 | same cell |
| $\eta < 0$ | j = j+1 | j = j-1 | j = j+1 | j = j-1 | same cell |
| $\xi < 0$ | k = k+1 | k = k-1 | k = k-1 | k = k+1 | same cell |
| $1-\xi-\eta-\zeta < 0$ | same cell | same cell | same cell | same cell | same cell |

(b)

Fig. 13. Cell index look-up tables for the (a) odd and (b) even tetrahedral decompositions

## ACKNOWLEDGEMENTS

We are very grateful to Steve Bryson for helping us install the tetrahedral method in the Virtual Windtunnel. We are also grateful to the following people for allowing us to use their data sets. Sungho Ko for the oscillating airfoil, Dennis Jespersen and Creon Levit for the tapered cylinder, and Neal Chaderjian for the delta wing. This work was supported by NASA under contract NAS 2-12961.

## REFERENCES

[1] D. Kenwright and D. Lane, "Optimization of Time-Dependent Particle Tracing Using Tetrahedral Decomposition", Proc. Visualization '95, IEEE CS Press, pp. 321-328, Atlanta, Georgia, October 1995.

[2] A. Globus, "A Software Model for Visualization of Large Unsteady 3-D CFD Results", AIAA 33rd Aerospace Sciences Meeting and Exhibit, AIAA 95-0115, Reno, Nevada, 1995.

[3] D. Lane, "Scientific Visualization of Large Scale Unsteady Fluid Flow", Scientific Visualization: Surveys, Methodologies and Techniques, IEEE CS Press, 1996.

[4] W. Merzkirch, Flow Visualization, McGraw-Hill, 1st Edition, pp. 1-25, 1979.

[5] M. Smith, W. Van Dalsem, F. Dougherty, and P. Buning, "Analysis and Visualization of Complex Unsteady Three-Dimensional Flows", AIAA 27th Aerospace Sciences Meeting, AIAA 89-0139, Reno, Nevada, 1989.

[6] C. Chow, R. Leben, and L. Gea, "Numerical Simulation of Streaklines in Unsteady Flows", AIAA 27th Aerospace Sciences Meeting, AIAA 89-0292, Reno, Nevada, 1989.

[7] D. Jespersen and C. Levit, "Numerical Simulation of Flow Past a Tapered Cylinder", AIAA Annual Aerospace Sciences Meeting, AIAA 91-0751, New York, 1991.

[8] D. Lane, "Visualization of Time-Dependent Flow Fields", Proc. Visualization '93, IEEE CS Press, pp. 32-38, San Jose, California, October, 1993.

[9] S. Bryson and C. Levit, "The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows", Proc. Visualization '91, IEEE CS Press, pp. 17-24, San

Diego, California, October, 1991.

[10] A. Sadarjoen, T. van Walsum, A. Hin, and F. Post, "Particle Tracing Algorithms for 3-D Curvilinear Grids", Proc. 5th Eurographics Workshop on Visualization in Scientific Computing, Rostock, Germany, 1994.

[11] P. Buning, "Sources of Error in the Graphical Analysis of CFD Results", Journal of Scientific Computing, vol. 3, no. 2, pp. 149-164, 1988.

[12] J. Hultquist, Interactive Numerical Flow Visualization Using Stream Surfaces, Ph.D. dissertation, pp. 53-91, University of North Carolina at Chapel Hill, May 1995.

[13] T. Pulliam, "Efficient Solution Methods for Navier-Stokes Equations", Proc. von Karman Institute for Fluid Dynamics Lecture Series, pp. 78, Belgium, January 1986.

[14] D. Darmofal and R. Haimes, "An Analysis of 3-D Particle Path Integration Algorithms", Proc. 1995 AIAA CFD Meeting, San Diego, California, 1995.

[15] P. Buning, "Numerical Algorithms in CFD Post-Processing", Computer Graphics and Flow Visualization in Computational Fluid Dynamics, Proc. von Karman Institute for Fluid Dynamics Lecture Series, 1989-07, September 1989.

[16] T. Chung, Finite Element Analysis in Fluid Dynamics, McGraw Hill, pp. 60-93, 1977.

[17] G. Dhatt and G. Touzot, The Finite Element Method Displayed, J. Wiley, pp. 110, 1984.

[18] S. Rao, The Finite Element Method in Engineering, Pergamon Press, pp. 121-135, 1982.

[19] W. Press et al., Numerical Recipes, Cambridge University Press, pp. 554-560, 1986.

[20] D. Darmofal and R. Haimes, "Visualization of 3-D Vector Fields: Variations on a Stream", AIAA 30th Aerospace Sciences Meeting and Exhibit, AIAA 92-0074, Reno, Nevada, January 1992.